J.R. Koza, F.H. Bennett, J.D. Lohn, F. Dunlap, D. Andre, M.A. Keane, ``Automated Synthesis of Computational Circuits using Genetic Programming," 1997 IEEE International Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, 1997, pp. 447--452.

# Automated Synthesis of Computational Circuits Using Genetic Programming

**John R. Koza**
Computer Science Dept.
258 Gates Building
Stanford University
Stanford, California 94305-9020
koza@cs.stanford.edu
http://www-cs-faculty.stanford.edu/~koza/

**Forrest H Bennett III**
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
fhb3@slip.net

**Jason Lohn**
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
jlohn7@leland.stanford.edu

**Frank Dunlap**
Dunlap Consulting
Palo Alto, California

**Martin A. Keane**
Martin Keane Inc.
5733 West Grover
Chicago, Illinois 60630
makeane@ix.netcom.com

**David Andre**
Computer Science Division
University of California
Berkeley, California
dandre@cs.berkeley.edu

**Abstract:** Analog electrical circuits that perform mathematical functions (e.g., cube root, square) are called *computational circuits*. Computational circuits are of special practical importance when the small number of required mathematical functions does not warrant converting an analog signal into a digital signal, performing the mathematical function in the digital domain, and then converting the result back to the analog domain. The design of computational circuits is difficult even for mundane mathematical functions and often relies on the clever exploitation of some aspect of the underlying device physics of the components. Moreover, implementation of each different mathematical function typically requires an entirely different clever insight. This paper demonstrates that computational circuits can be designed without such problem-specific insights using a single uniform approach involving genetic programming. Both the circuit topology and the sizing of all circuit components are created by genetic programming. This uniform approach to the automated synthesis of computational circuits is illustrated by evolving circuits that perform the cube root function (for which no circuit was found in the published literature) as well as for the square root, square, and cube functions.

## 1. Introduction

Analog electrical circuits that perform mathematical functions (e.g., cube root, square) are called *computational circuits*. Computational circuits are of special practical importance when the small number of required mathematical functions does not warrant converting an analog signal into a digital signal, performing the mathematical function in the digital domain, and then converting the result back to the analog domain. The design of computational circuits is difficult even for mundane mathematical functions and often relies on clever exploitation of some aspect of the underlying physics of the components. Each function usually requires a different clever insight (Gilbert 1968, Sheingold 1976, Babanezhad and Temes 1986).

This paper demonstrates that computational circuits can be designed by means of a single uniform approach using genetic programming. Both the circuit topology and the sizing of all circuit components are created by genetic programming. This uniform approach to the automated synthesis of computational circuits is illustrated by evolving a circuit for the cube root, square root, square, and cube functions.

The problem of circuit synthesis involves designing an electrical circuit that satisfies user-specified design goals. The design of analog circuits and mixed analog-digital circuits has not proved to be amenable to automation (Rutenbar 1993). Thompson (1996) used a genetic algorithm to evolve a frequency discriminator on a Xilinx 6216 reconfigurable gate array in analog mode. CMOS operational amplifier (op amp) circuits have been designed using a modified version of the genetic algorithm (Kruiskamp and Leenaerts 1995); however, the topology of each op amp was one of 24 pre-selected topologies based on the conventional human-designed op amp stages. Evolvable digital hardware (Higuchi et al. 1993; Sanchez and Tomassini 1996) offers a potential approach to automated synthesis of digital circuits.

## 2. Evolution of Circuits

Genetic programming is an extension of John Holland's genetic algorithm (1975) in which the population consists of computer programs of varying sizes and shapes (Koza 1992, 1994a, 1994b; Koza and Rice 1992). Recent research on genetic programming is described in Kinnear (1994), Angeline and Kinnear (1996), and Koza, Goldberg, Fogel, and Riolo (1996).

Genetic programming ordinarily evolves computer programs that are represented as rooted, point-labeled trees with ordered branches. Genetic programming can be applied to circuits if a mapping is established between the program trees found in genetic programming and the line-labeled cyclic graphs germane to circuits.

Developmental biology suggests a way to map program trees into circuits. Using these principles, Gruau (1996) evolved neural networks using genetic programming. The starting point of the growth process used herein is a very simple embryonic electrical circuit. The embryonic circuit contains certain fixed parts appropriate to the problem at hand and certain wires that are capable of subsequent modification. An electrical circuit is progressively developed by applying the functions in a circuit-constructing program tree to the modifiable wires of the embryonic circuit (and, at each later step of the development, to both the modifiable wires and the other components of the developing circuit).

The functions in the circuit-constructing program trees include (1) connection-modifying functions that modify the topology of the circuit, (2) component-creating functions that insert components into the circuit, (3) arithmetic-performing functions that appear in arithmetic-performing subtrees as argument(s) to the component-creating functions and that specify the numerical value of the component, and (4) calls to automatically defined functions that appear in function-defining branches.

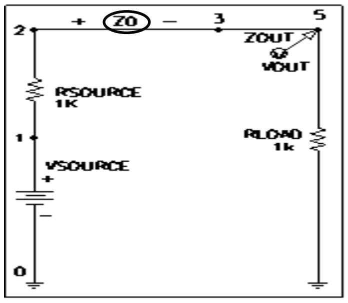

**Figure 1 Embryonic circuit.** The developmental process for converting a program tree into an electrical circuit begins with an embryonic circuit. Figure 1 shows a one-input, one-output embryonic circuit. This embryo contains a voltage source VSOURCE connected to nodes 0 (ground) and 1, a fixed source resistor RSOURCE between nodes 1 and 2, a modifiable wire Z0 between nodes 2 and 3, a fixed isolating wire ZOUT between nodes 3 and 5, an output point (voltage probe) VOUT at node 5, and a fixed load resistor RLOAD between nodes 5 and ground. Only the modifiable wire Z0 is subject to modification during the developmental process.

Each branch of the program tree is created in accordance with a constrained syntactic structure. Branches are composed from construction-continuing subtrees that continue the developmental process and arithmetic-performing subtrees that determine the numerical value of components. Each circuit-constructing program tree in the population contains component-creating functions and connection-modifying functions. Each connection-modifying function in a program tree points to an associated highlighted component and modifies the topology of the developing circuit. Connection-modifying functions have one or more construction-continuing subtrees, but no arithmetic-performing subtrees. Component-creating functions have one construction-continuing subtree and typically have one arithmetic-performing subtree. This constrained syntactic structure is preserved by using structure-preserving crossover with point typing (Koza 1994a).

Component-creating functions insert a component into the developing circuit and assign component value(s) to the component. Each component-creating function has a writing head that points to an associated highlighted component in the developing circuit and modifies the highlighted component in a specified way. The construction-continuing subtree of each component-creating function points to a successor function or terminal in the circuit-constructing program tree.

The arithmetic-performing subtree of a component-creating function consists of a composition of arithmetic functions and random constants that specify, after interpretation, the numerical value of a component.

Space does not permit a detailed description of each function herein.

Various electrical circuits have been designed using genetic programming, including lowpass filters (Koza, Bennett, Andre, and Keane 1996a, 1996b), crossover (woofer and tweeter) filters (Koza, Bennett, Andre, and Keane 1996c), asymmetric bandpass filters (Koza, Bennett, Andre, and Keane 1996d), and a 60 dB operational amplifier (Bennett, Koza, Andre, and Keane 1996), and the use of automatically defined functions and architecture-altering operations for creating useful electrical subcircuits (Koza, Andre, Bennett, and Keane 1996).

## 3. Preparatory Steps

Before applying genetic programming to a problem of circuit synthesis, the user must perform seven major preparatory steps, namely (1) identifying the embryonic circuit that is suitable for the problem, (2) determining the architecture of the overall circuit-constructing program trees, (3) identifying the terminals of the to-be-evolved programs, (4) identifying the primitive functions contained in the to-be-evolved programs, (5) creating the fitness measure, (6) choosing certain control parameters, and (7) determining the termination criterion and method of result designation.

The one-input, one-output embryo of figure 1 (with one modifiable wire Z0) is suitable for the synthesis of computational circuits.

Since the embryonic circuit has one modifiable wire, there is one result-producing branch in each circuit-constructing program tree.

The function set, $F_{\text{ccs}}$, for the construction-continuing subtrees is

$F_{\text{ccs}}$ = {R, SERIES, PSS, PSL, FLIP, NOP, NEW_T_GND_0, NEW_T_GND_1, NEW_T_POS_0, NEW_T_POS_1, NEW_T_NEG_0, NEW_T_NEG_1, PAIR_CONNECT_0, PAIR_CONNECT_1, Q_D_NPN, Q_D_PNP, Q_3_NPN0, ..., Q_3_NPN11, Q_3_PNP0, ..., Q_3_PNP11, Q_POS_COLL_NPN, Q_GND_EMIT_NPN, Q_NEG_EMIT_NPN, Q_GND_EMIT_PNP, Q_POS_EMIT_PNP, Q_NEG_COLL_PNP}

The terminal set, $T$ccs, for the construction-continuing subtree is

$T$ccs = {END, SAFE_CUT}.

The function set, $F$aps, for each arithmetic-performing subtree is

$F$aps = {+, -}.

The terminal set, $T$aps, for each arithmetic-performing subtree is

$T$aps = {$\Re$}.

$\Re$ represents random constants from –1.0 to +1.0.

SPICE's default npn and pnp transistor model parameters were used.

The evaluation of fitness for each individual circuit-constructing program tree in the population begins with its execution. This execution applies the functions in the program tree to the very simple embryonic circuit, thereby developing the embryonic circuit into a fully developed circuit. A netlist describing the circuit is then created. The netlist identifies each component of the circuit, the nodes to which that component is connected, and the value of that component. The circuit is then simulated to determine its behavior. The 217,000-line SPICE simulator was modified to run as a submodule within the genetic programming system. SPICE is a large family of programs written over several decades at the University of California at Berkeley for the simulation of analog, digital, and mixed analog/digital electrical circuits (Quarles et al. 1994). The input to a SPICE simulation consists of a netlist describing the circuit to be analyzed and certain commands that instruct SPICE as to the type of analysis to be performed and output to be produced.

The fitness measure is customized to each particular desired computational circuit. For example, for the cube root circuit, the target voltage is the cube root of the input voltage. The SPICE simulator is requested to perform a DC sweep analysis at 21 equidistant voltages between –250 mV and +250 mV for the cube root, square, and cube functions (and 0 mV to +500 mV for the square root function). Fitness is the sum, over these 21 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit at the probe point VOUT at node 5 and the target value for voltage. The smaller the value of fitness, the better.

The fitness measure does not penalize output voltages that perfectly match the target voltages; it slightly penalizes every acceptable deviation from the target voltage; and it heavily penalizes every unacceptable deviation. If the output voltage is within 1% of the target voltage value for a particular fitness case, the absolute value of the deviation is weighted by 1 for that fitness case. If the output voltage is not within 1% of the target voltage value, the deviation is weighted by 10 for that fitness case. This arrangement reflects the fact that a deviation of 1% from the ideal voltage is acceptable, but greater deviations are not.

The population size, $M$, was 640,000. The percentage of genetic operations on each generation was 89% one-offspring crossovers, 10% reproductions, and 1% mutations. The architecture-altering operations were not used on this problem. Since only one result-producing branch was used in the embryo for this problem, the maximum size, $H_{r\text{pb}}$, for the result-producing branch was 600 points. The other parameters for controlling the runs of genetic programming were the default values specified in Koza 1994a (appendix D).

This problem was run on a medium-grained parallel Parsytec computer system consisting of 64 80 MHz Power PC 601 processors arranged in a toroidal mesh with a host PC Pentium type computer. The distributed genetic algorithm was used. On each generation, four boatloads of emigrants, each consisting of $B$ = 2% (the migration rate) of each node's subpopulation (selected on the basis of fitness) were dispatched to each of the four toroidally adjacent processing nodes. See Andre and Koza 1996 for details.

## 4. Results

### 4.1. Cube Root Circuit

The goal here is to evolve an analog electrical circuit whose output is the cube root of its input.

The worst individual program trees from generation 0 create circuits that are so pathological that SPICE is incapable of simulating them.

The best circuit from generation 0 (figure 5) achieves a fitness of 77.7 and has two transistors, no diodes, and one resistor (in addition to the source and load resistors in the embryo). Figure 2 compares the output produced by the best circuit from generation 0 with the target (i.e., the cube root of the input voltage). As can be seen, the output resembles the target only in that it has a positive slope.

Fitness improves as the evolutionary process proceeds from generation to generation. The best circuit from generation 17 (figure 6) achieves a fitness of 26.7 and has 13 transistors, three diodes, no capacitors, and two resistors (in addition to the source and load resistors in the embryo). Figure 3 compares the output produced by the best circuit from generation 17 with the target (i.e., the cube root of the input voltage).

The best circuit from generation 60 (figure 7) achieves a fitness of 1.68 and has 36 transistors, two diodes, no capacitors, and 12 resistors. Figure 4 shows that the output of this circuit is virtually the same as the target (i.e., the cube root of the input).

### 4.2. Square Root, Squaring, and Cubing Circuits

The design of several other computational circuits have been evolved using genetic programming. The best-of-run circuit (figure 8) for the problem of designing a square root circuit has 39 transistors, seven diodes, and 18 resistors. The best-of-run circuit (figure 9) for the problem of designing a squaring circuit has 33 transistors, five diodes, and one resistor. The best-of-run circuit (figure 10) for the problem of designing a cubing circuit has 30 transistors, five diodes, and 21 resistors.

## 5. Conclusion

We evolved circuits that perform the cube root, square root, square, and cube functions.

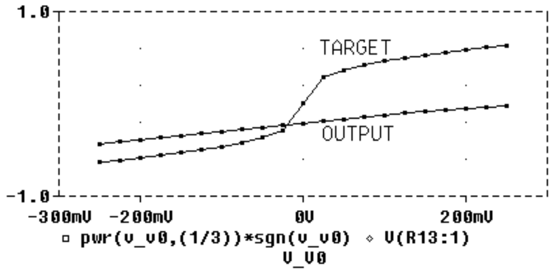

**Figure 2 Comparison for generation 0.**

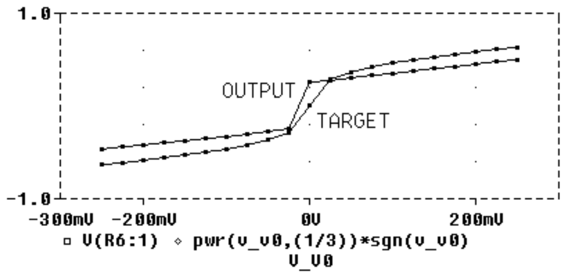

**Figure 3 Comparison for generation 17.**

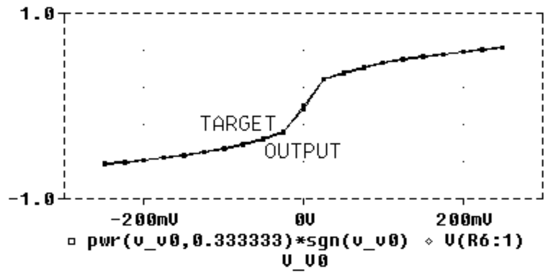

**Figure 4 Comparison for generation 60.**

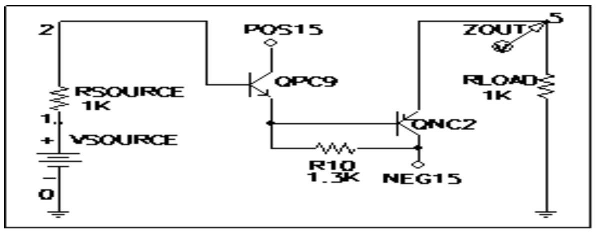

**Figure 5 Best circuit from generation 0.**

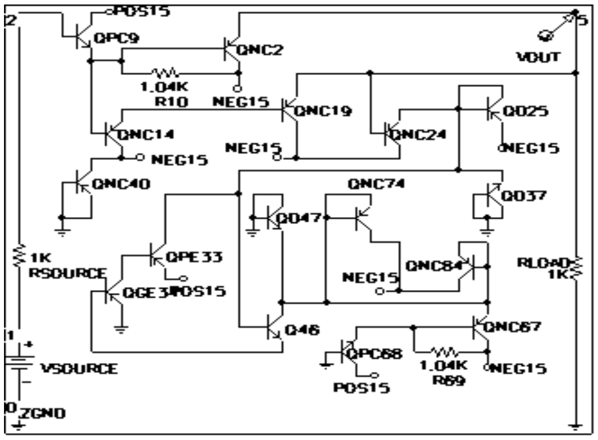

**Figure 6 Best circuit from generation 17.**

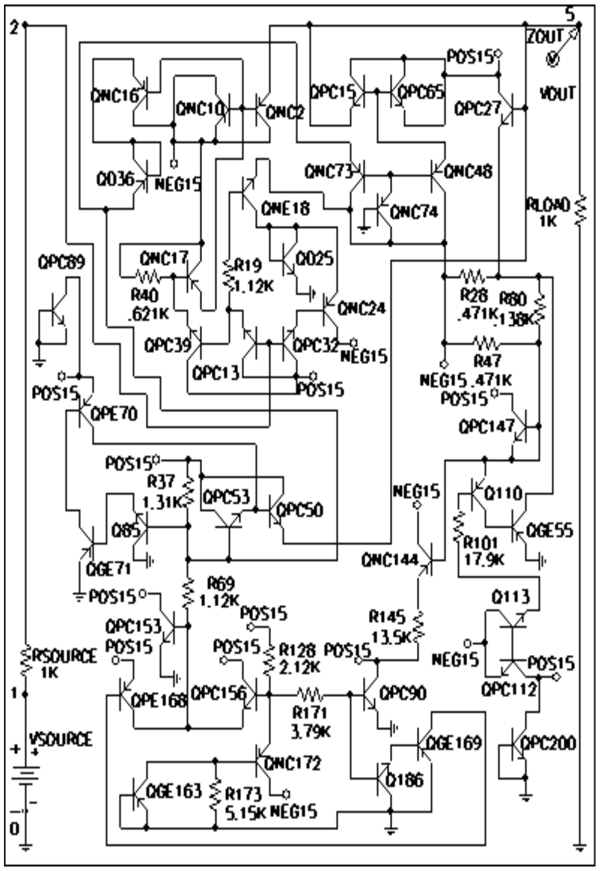

**Figure 7 Best circuit from generation 60.**

## 6. Acknowledgments

Dr. Joseph Babanezhad provided helpful information to Frank Dunlap concerning this paper. Simon Handley made helpful comments on this paper.

## References

Andre, David and Koza, John R. 1996. Parallel genetic programming: A scalable implementation using the transputer architecture. In Angeline, P. J. and Kinnear, K. E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge: MIT Press.

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge, MA: The MIT Press.

Babanezhad, J. N. and Temes, G. C. 1986. Analog MOS Computational Circuits. *Proceedings of the IEEE Circuits and System International Symposium*. Pages 1156–1160.

Bennett III, Forrest H, Koza, John R., Andre, David, and Keane, Martin A. 1996. Evolution of a 60 Decibel op amp using genetic programming. In *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware*. Lecture Notes in Computer Science. Berlin: Springer-Verlag.

Gilbert, Barrie. 1968. A precise four-quadrant multiplier with subnanosecond response. *IEEE Journal of Solid-State Circuits*. Volume SC-3. Number 4. December 1968. Pages 365–373.

Gruau, Frederic. 1996. Artificial cellular development in optimization and compilation. In Sanchez, Eduardo and Tomassini, Marco (editors). 1996. *Towards Evolvable Hardware.* Lecture Notes in Computer Science, Volume 1062. Berlin: Springer-Verlag. Pages 48–75.

Higuchi, Tetsuya, Niwa, Tatsuya, Tanaka, Toshio, Iba, Hitoshi, de Garis, Hugo, and Furuya, Tatsumi. 1993. In Meyer, Jean-Arcady, Roitblat, Herbert L. and Wilson, Stewart W. (editors). *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: The MIT Press. 1993. Pages 417–424.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Koza, John R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation.* Cambridge, MA: MIT Press.

Koza, John R. 1995. Gene duplication to enable genetic programming to concurrently evolve both the architecture and work-performing steps of a computer program. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann. Pages 734–740.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin A. 1996. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996a. Toward evolution of electronic animals using genetic programming. *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996b. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In Gero, John S. and Sudweeks, Fay (editors). *Artificial Intelligence in Design '96*. Dordrecht: Kluwer. Pages 151-170.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996c. Four problems for which a computer program evolved by genetic programming is competitive with human performance. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. IEEE Press. Pages 1–10.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996d. Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996e. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware.* Lecture Notes in Computer Science. Berlin: Springer-Verlag.

Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: The MIT Press.

Koza, John R., and Rice, J. P. 1992. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press.

Kruiskamp, Marinum Wilhelmus and Leenaerts, Domine. 1995. DARWIN: CMOS opamp synthesis by means of a genetic algorithm. *Proceedings of the 32nd Design Automation Conference*. New York: Association for Computing Machinery. 433–438.

Quarles, Thomas, Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A. 1994. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA. March 1994.

Rutenbar, R. A. 1993. Analog design automation: Where are we? Where are we going? *Proceedings of the l5th IEEE CICC*. New York: IEEE. 13.1.1-13.1.8.

Sanchez, Eduardo and Tomassini, Marco (editors). 1996. *Towards Evolvable Hardware.* Lecture Notes in Computer Science, Volume 1062. Berlin: Springer-Verlag.

Sheingold, Daniel H. (editor). 1976. *Nonlinear Circuits Handbook*. Norwood, MA: Analog Devices.

Thompson, Adrian. 1996. Silicon evolution. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

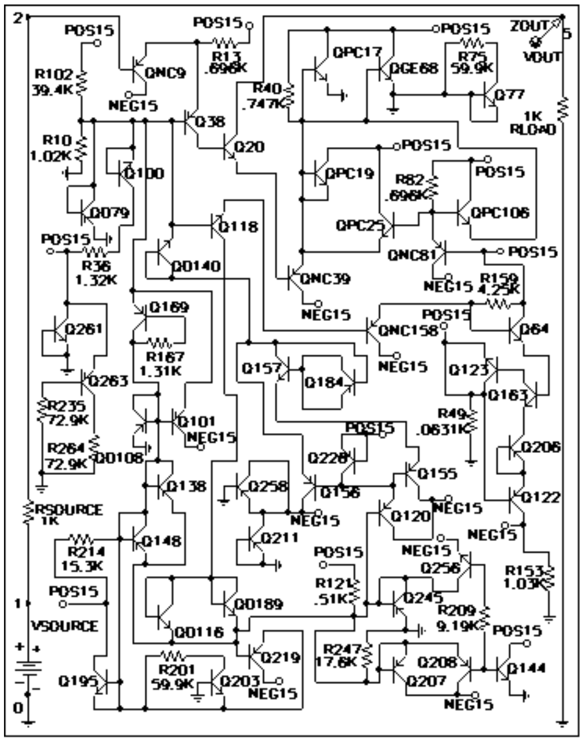

**Figure 8 Evolved square root circuit.**

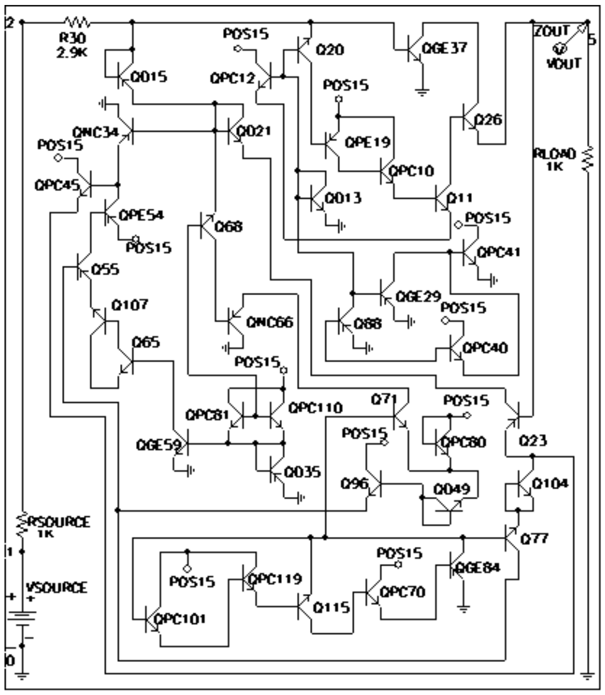

**Figure 9 Evolved squaring circuit.**

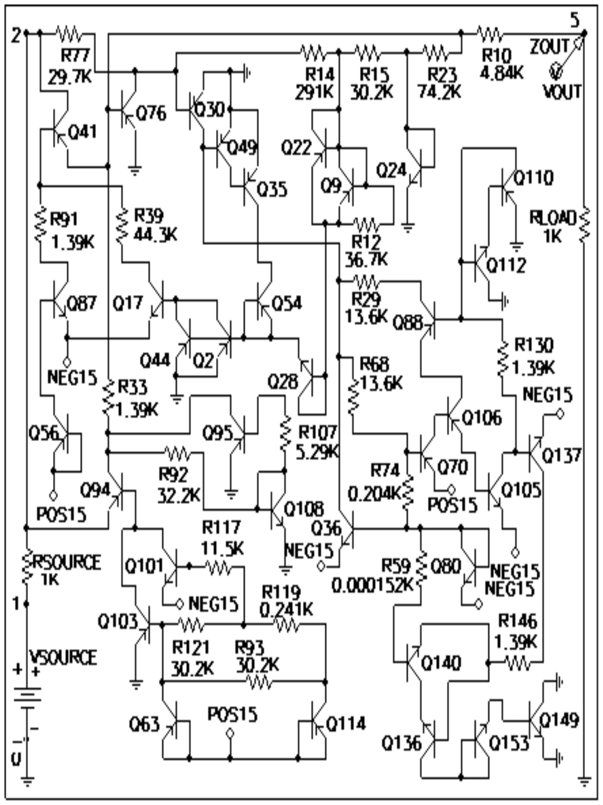

**Figure 10 Evolved cubing circuit.**

Version 2 – **Camera-Ready Version** – Submitted January 24, 1997 to IEEE International Conference on Evolutionary Computation (ICEC-97) to be held in Indianapolis on April 13 – 16 (Sunday – Thursday), 1997.

# Automated Synthesis of Computational Circuits Using Genetic Programming

**John R. Koza**
Computer Science Dept.
258 Gates Building
Stanford University
Stanford, California 94305-9020
koza@cs.stanford.edu
http://www-cs-faculty.stanford.edu/~koza/

**Forrest H Bennett III**
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
fhb3@slip.net

**Jason Lohn**
Visiting Scholar
Computer Science Dept.
Stanford University
Stanford, California 94305
jlohn7@leland.stanford.edu

**Frank Dunlap**
Dunlap Consulting
Palo Alto, California

**Martin A. Keane**
Martin Keane Inc.
5733 West Grover
Chicago, Illinois 60630
makeane@ix.netcom.com

**David Andre**
Computer Science Dept.
University of California
Berkeley, California
dandre@cs.berkeley.edu